Neural Abstractive Text Summarization and Fake News Detection

Soheil Esmaeilzadeh
soes@stanford.edu

Gao Xian Peh
gxpeh@stanford.edu

Angela Xu
aaxu@stanford.edu

Abstract

In this work, we study abstractive text summarization by exploring different models such as LSTM-encoder-decoder with attention, pointer-generator networks, coverage mechanisms, and transformers. Upon extensive and careful hyperparameter tuning we compare the proposed architectures against each other for the abstractive text summarization task. Finally, as an extension of our work, we apply our text summarization model as a feature extractor for a fake news detection task where the news articles prior to classification will be summarized and the results are compared against the classification using only the original news text.

keywords: abstractive text summarization, pointer-generator, coverage mechanism, transformers, fake news detection

1 Introduction

In simple words, text summarization is the task of creating a summary for a large piece of text. Generating *meaningful* summaries of long texts is of great importance in many different areas such as medical, educational, media, social, and etc., where the summary needs to contain the main contextual aspects of the text while reducing the amount of unnecessary information.

In general, text summarization can be classified into two main groups: *extractive summarization* and *abstractive summarization* [Allahyari et al.]. Extractive summarization creates summaries by synthesizing salient phrases from the full text verbatim [Dorr et al., 2007, Nallapati et al., 2016a], however, abstractive summarization creates an internal semantic representation of the text. Unlike extractive summarization which concatenates sentences taken explicitly from the source text, abstractive text summarization paraphrases the text in a way that it is closer to the human's style of summarization and this makes abstractive text summarization a challenging yet preferable approach [Khatri et al., 2018, Gao et al., 2018].

Decent quality summaries using abstractive approaches were only obtained in the past few years by applying the sequence-to-sequence endoder-decoder architectures with attention mechanisms common in machine translation tasks to summarization [Bahdanau et al., Nallapati et al., 2016b] however only focused on short input texts. Subsequent works attempted to perform the abstractive summarization task on longer input texts, however, appearance of unknown words and repetitions adversely affected the outcome of the summarization tasks [Moritz et al.].

In this work, we focus on abstractive text summarization as a more robust approach compared to its counterpart (i.e. extractive summarization) and explore recent advancements in the state-of-the-art natural language models for abstractive text summarization. The input of our natural language model is a single document or article and the output of it is a combination of a few sentences that summarize the content of the input document in a meaningful manner. In addition to the main goal of this work, after exploring the natural language models for abstractive text summarization, we use the summarization model as a feature building module for fake news detection and news headline generation, and show the effect of summarization on fake news detection.
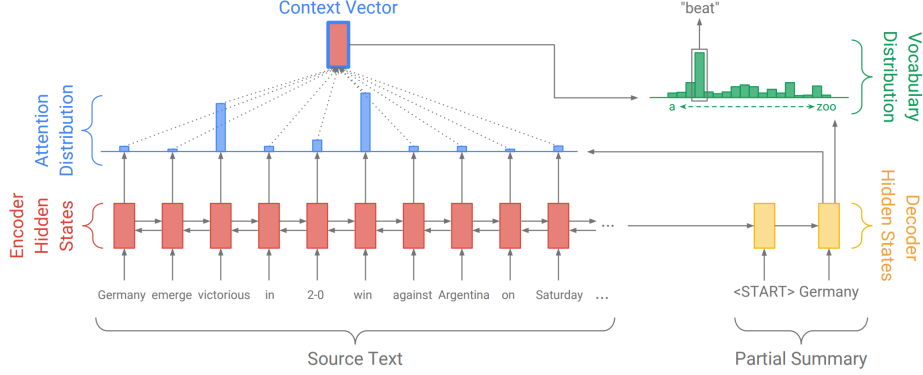
Figure 1: Baseline sequence-to-sequence model's architecture with attention [See et al., 2017]

2 Approaches

2.1 Baseline Model

In this work, as the baseline model we consider an LSTM Encoder-Decoder architecture with attention as shown in Figure 1.

Sequence-to-Sequence Encoder-Decoder: The sequence-to-sequence framework consists of a Recurrent Neural Network (RNN) encoder and an RNN decoder. The RNN encoder as a single-layer bidirectional Long-short-term-memory (LSTM) unit reads in the input sequence token by token and produces a sequence of encoder's hidden states h_i that encode or represent the input. The RNN decoder as a single-layer unidirectional LSTM generates the decoder's hidden states s_t one by one which produces the output sequence as the summary.

Attention Mechanism: In the attention mechanism, an attention distribution a^t is calculated as a probability distribution over the words in the source text that helps the decoder decide which source words to concentrate on when it generates the next word. The attention distribution a^t is calculated for each decoder timestep t as:

$$e_i^t = v^T \tanh(W_h h_i + W s_t + b_{attn}), \tag{1a}$$

$$a^t = \mathrm{softmax}(e^t), \tag{1b}$$

where v, W_h, W_s, b_{attn} are learnable parameters. On each decoder's step, attention weights a_i^t, which are part of the a^t distribution for the source words are computed. An attention weight represents the amount of attention that should be paid to a certain source word in order to generate an output word (decoder state) in the decoder. The attention distribution is used to compute a weighted sum of the encoder hidden states, known as the context vector h_t^*, which represents what has been read from the source for this step, and can be calculated as:

$$h_t^* = \sum_i a_i^t h_i . \tag{2}$$

The context vector along with the decoder's state are then used to calculate the vocabulary distribution P_{vocab}, which provides a final distribution for predicting words w as:

$$P_{vocab} = \mathrm{softmax}(V'(V[s_t, h_t^*] + b) + b'), \tag{3a}$$

$$P(w) = P_{vocab}(w), \tag{3b}$$

where V, V', b, and b' are learnable parameters. Subsequently, we calculate the loss for the timestep t as the negative log-likelihood of the target word w_t^* as:

$$loss_t = -\log P(w_t^*) . \tag{4}$$

The overall loss for the whole sequence is the average of the loss at each time step (i.e. $loss_t$) as:

$$loss = \frac{1}{T} \sum_{t=0}^{T} loss_t . \tag{5}$$

Baseline Model's Problems: Some problems are associated with the baseline model proposed in section 2.1. One problem is the model's tendency to reproduce factual details inaccurately, this happens specially when an uncommon word that exists in the vocabulary is replaced with a more common word. Another problem with the baseline model is that during summary generation it repeats the already generated parts of the summary. Lastly, the baseline is unable to handle out-of-vocabulary words (OOV). In general, it is hard for the sequence-to-sequence-with-attention model to copy source words as well as to retain longer-term information in the decoder state, which leads to the aforementioned issues. See et al. [2017] proposed a so called *pointer-generator* network that also includes a coverage mechanism in order to address these problems by combining both context extraction (pointing) and context abstraction (generating). We revisit the model proposed by See et al. [2017] in the following and as well compare it with a transformer based model proposed by Miyagishima et al. [2014] for machine translation tasks, and finally use it as a feature generation mechanism for fake news classification.
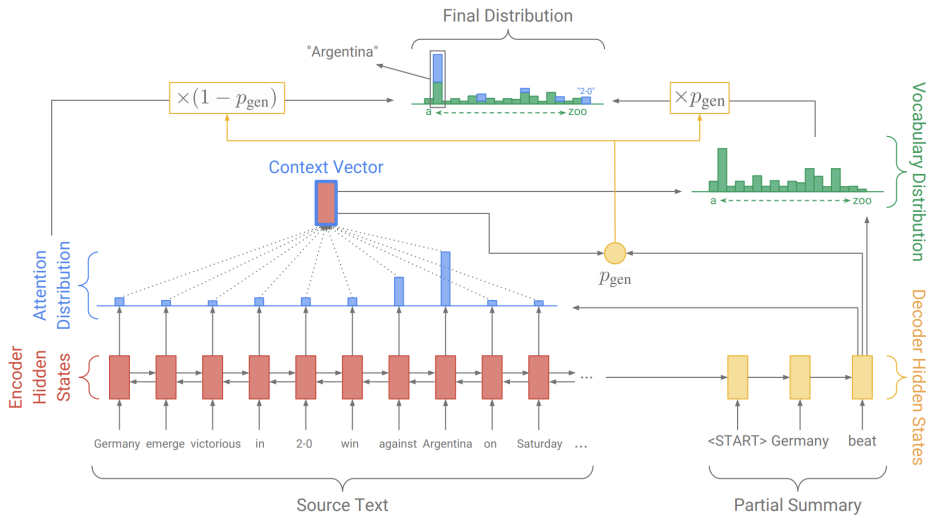
2.2 Pointer-Generator Network

Figure 2: Pointer-generator model's architecture [See et al., 2017]

Pointer-Generator Mechanism: Pointer-generator is a hybrid network that chooses during training and test whether to copy words from the source via pointing or to generate words from a fixed vocabulary set. Figure 2 shows the architecture for the pointer-generator mechanism where the decoder part is modified compared to Figure 1. In Figure 1, the baseline model, only an attention distribution and a vocabulary distribution are calculated. However, here in the pointer-generator network a generation probability p_{gen}, which is a scalar value between 0 and 1 is also calculated which represents the probability of generating a word from the vocabulary, versus copying a word from the source text. The generation probability p_{gen} weights and combines the vocabulary distribution $\mathrm{P}_{\text{vocab}}$ (used for generating) and the attention distribution a (used for pointing to source words ω_i) into the final distribution $\mathrm{P}_{\text{final}}$ as:

$$\mathrm{P}_{\text{final}}(w) = p_{\text{gen}}\mathrm{P}_{\text{vocab}}(w) + (1 - p_{\text{gen}}) \sum_{i:w_i=w} a_i \,. \tag{6}$$

Based on Equation (6), the probability of producing word ω is equal to the probability of generating it from the vocabulary multiplied by the generation probability plus the probability of pointing to it anywhere it appears in the source text multiplied by the copying probability. Compared to the LSTM Encoder-Decoder model with attention as baseline in section 2.1, the pointer-generator network makes it easy to copy words from the source text by putting sufficiently large attention on the relevant word. It also is able to copy out-of-vocabulary words from the source text, enabling the model to handle unseen words while allowing to use a smaller vocabulary, leading to less computation and storage space. The pointer-generator model is also faster to train, as it requires fewer training iterations to achieve the same performance as the baseline model in section 2.1.
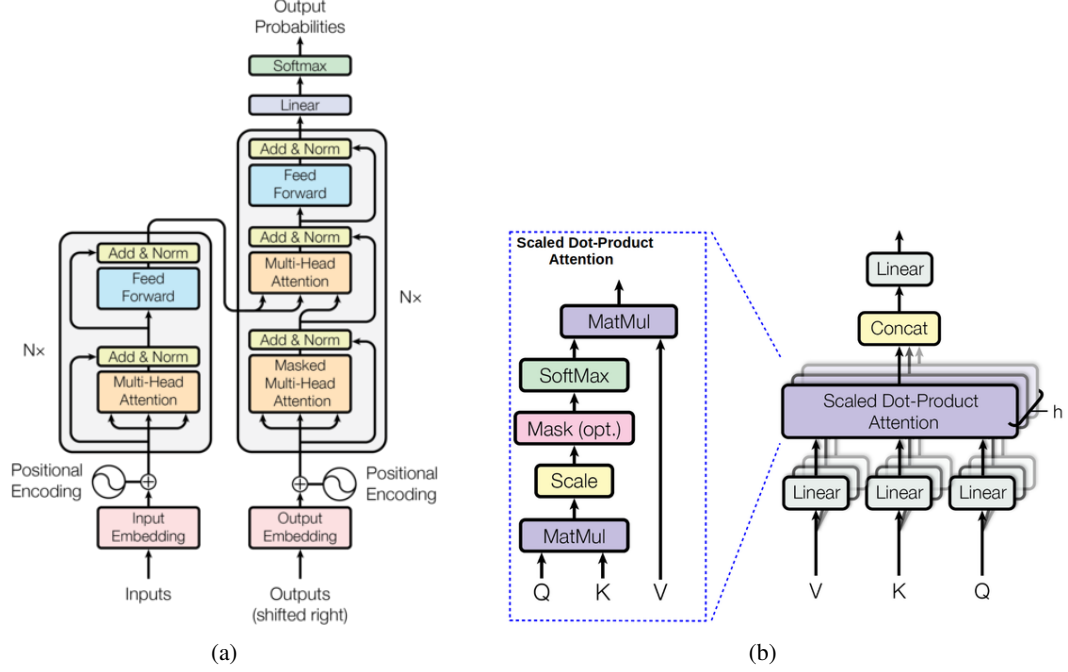
3

Figure 3: (a). The Transformer - model architecture, (b). (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel - [Miyagishima et al., 2014]

Coverage Mechanism: To reduce the repetition during summarization as a common issue with sequence-to-sequence models mentioned in section 2.1, we apply the coverage mechanism, first proposed by Tu and Lu [2016] and adapted by See et al. [2017]. The coverage mechanism keeps track of a coverage vector, computed as the sum of attention distributions over previous decoder time steps. This coverage vector is incorporated into the attention mechanism and represents the degree of coverage that words in the source text have received from the attention mechanism so far. Thus, by maintaining this coverage vector, which represents a cumulative attention, the model avoids attending to any word that has already been covered and used for summarization.

On each timestep t of the decoder, the coverage vector c^t is the sum of all the attention distributions $a^{t'}$ so far as:

$$c^t = \sum_{t'=0}^{t-1} a^{t'} .\tag{7}$$

This coverage vector also contributes to computing the attention mechanism described in the previous section, so that Equation (1a) becomes:

$$e_i^t = v^T \tanh(W_h h_i + W s_t + w_c c_i^t + b_{attn}) .\tag{8}$$

Intuitively, this informs the attention mechanism's current timestep about the previous attention information which is captured in c^t, thus preventing repeated attention to the same source words. To further discourage repetition, See et al. [2017] penalizes repeatedly attending to the same parts of the source text by defining a coverage loss and adding it to the primary loss function in Equation (4). This extra coverage loss term penalizes any overlap between the coverage vector c^t and the new attention distribution a^t as:

$$\text{covloss}_t = \sum_i \min(a_i^t, c_i^t) .\tag{9}$$

Finally the total loss becomes: $\text{loss} = \frac{1}{T} \sum_{t=0}^{T} (\text{loss}_t + \text{covloss}_t)$. For the aforementioned models we have consulted the Gihub repositories referenced at the end of this report.

4

2.3 Transformers

In this part, we revisit the *transformers network* proposed by Miyagishima et al. [2014] for machine translation, and investigate its performance on abstractive text summarization on our dataset. In the transformer model, the encoder maps an input sequence of symbol representations as $\mathbf{x} = (x_1, ..., x_n)$ to a sequence of continuous representations as $\mathbf{z} = (z_1, ..., z_n)$. Given $\mathbf{z}$, the decoder then generates an output sequence as $\mathbf{y} = (y_1, ..., y_n)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. The Transforthmer follows this overall architecture using stacked self-attention and point-wise fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 3, respectively. The encoder part of this architecture is mainly a stack of some identical layers where each one has two sublayers. The first is a multi-head self-attention mechanism, and the second is a simple, position wise fully connected feed-forward network. The decoder is also composed of a stack of identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. In the transformer architecture a variation of attention mechanism called *Scaled Dot-Product Attention* is used where the input consists of queries and keys of dimension d_k, and values of dimension d_v. The dot products of the query with all keys is calculated, then divided by $\sqrt{d_k}$, the result goes through a softmax function to obtain the weights on the values. In practice the attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V, where the matrix of output can be calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \ . \tag{10}$$

In the proposed transformer model by Miyagishima et al. [2014] instead of performing a single attention function they linearly project the queries, keys, and values different times with different learned linear projections and that way they build a *multi-head attention*. On each of the projected versions of queries, keys, and values they then perform the attention function in parallel, yielding multi-dimensional output values which are concatenated and once again projected (Figure 3b). For the transformer model we have consulted the Gihub repositories referenced at the end of this report.

3 Experiments

3.1 Dataset Overview & Preprocessing

To train our summarization models, we use the CNN-Dailymail dataset, a collection of news articles and interviews that have been published on the two popular news websites CNN.com and Dailymail.com. Like the common styles on newspapers and journals, each article contains 3-4 highlighted sections that together form the summary of the whole article. The raw dataset includes the text contents of web-pages saved in separate HTML files [Chen et al., Wang]. We use the CNN and Dailymail dataset provided by DeepMind. Our dataset is split in 92%, 4.2%, 3.8% between training, dev, and test set respectively leading to 287,200 training pairs, 13,360 validation pairs, and 11,400 test pairs. There is an average of 781 tokens per news article. Each reference summary contains 3.75 sentences and 56 tokens on average.
We preprocess the dataset and convert the characters all to lower case. We use the Stanford CoreNLP library to tokenize the input articles and their corresponding reference summaries and to add paragraph and sentence start and end markers as $< p >$, $< /p >$ and $< s >$, $< /s >$ respectively. In addition, we have tried limiting our vocabulary size to 150k and 50k.

3.2 Evaluation Metric

We evaluate our models with the standard ROUGE (Recall-Oriented Understudy for Gisting Evaluation) score, a measure of the amount of overlap between the system-generated and the reference summaries (Lin and Rey [2001]). We report the F1, precision, and recall scores for ROUGE-1, ROUGE-2, and ROUGE-L, which measure respectively the word-overlap, bigram-overlap, and longest common sequence between the system-generated and reference summaries. The ROUGE recall and precision for summarization task can be calculated as:

$$\text{ROUGE recall} = \frac{\text{number of overlapping words}}{\text{total words in reference summary}} \ , \tag{11a}$$

$$\text{ROUGE precision} = \frac{\text{number of overlapping words}}{\text{total words in system summary}}, \tag{11b}$$

where the system summary refers to the summary generated by a summarization model. Using precision, it's possible to measure essentially how much of the system summary was in fact relevant or needed, and using recall ROUGE it's possible to measure how much of the reference summary is the summarization model generating. In terms of measuring the overlapping words in Equations (11a) and (11b), considering the overlap of *unigrams*, or *bigrams*, or *longest common sequence* leads to ROUGE-1, ROUGE-2, and ROUGE-L scores respectively for precision and recall.

3.3 Experimental Details & Results and Analysis

Text Summarization

In this work, we investigate the performance of the summarization models presented in section 2 namely: (1). LSTM encoder-decoder with only attention mechanism (baseline), (2). LSTM encoder-decoder with attention and pointer-generator mechanisms, (3). LSTM encoder-decoder with attention, pointer-generator, and coverage mechanisms, and (4). transformers. Table 1 shows the ROUGE-1, ROUGE-2, and ROUGE-L scores for the four different models that have been trained on the summarization dataset. We have trained the models upon hyperparameter tuning using Adagrad optimizer for 340 iterations (19 epochs). Our training results outperform the similar ones presented by See et al. [2017] for cases [1] and [2], and are very close in case [3].

| Model | ROUGE | | | | | | | | |
| | 1 | | | 2 | | | L | | |
	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall
[1]	35.68	44.07	31.95	14.21	17.66	12.87	30.56	41.02	29.67
[2]	38.47	43.02	36.98	16.33	18.68	15.94	33.37	39.60	33.99
[3]	38.97	42.71	38.21	16.81	18.12	16.22	35.41	38.63	35.04
[4]	36.55	43.33	34.50	15.21	17.92	13.89	31.19	40.38	31.54

Table 1: [1]. LSTM encoder-decoder with only attention mechanism (baseline), [2]. LSTM encoder-decoder with attention and pointer-generator mechanisms, [3]. LSTM encoder-decoder with attention, pointer-generator, and coverage mechanisms, and [4]. transformers
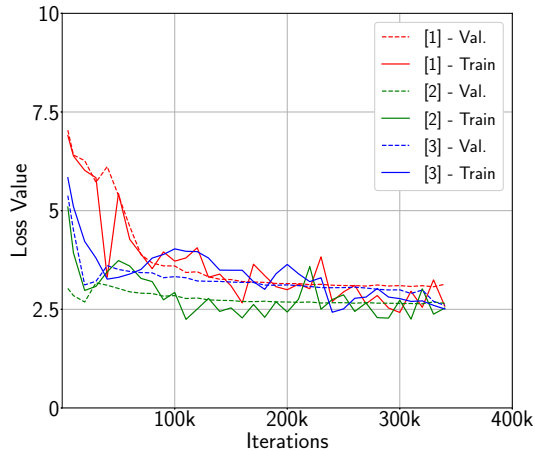

Figure 4: Validation and training loss values v.s. the number of iterations for summarization models

Figure 4 shows the loss on the training set and validation set for as a function of number of iterations for the summarization models for 340,000 iterations (19 epochs). The results of summarization are compared for one case v.s. its ground truth for the three summarization models in Table 2. As it can be seen the summary generated by model [1] contains $< unk >$ instead of the word *mysak* in the original summary. However, due to having attention and the pointer-generator mechanism model [2] has replaced the $< unk >$ with the proper word from the source text. However, summary of model

Reference Summary	once a super typhoon , maysak is now a tropical storm with 70 mph winds . it could still cause flooding , landslides and other problems in the philippines .
Summary by model [1]	[UNK] gained super typhoon status thanks to its sustained 150 mph winds. it 's now classified as a tropical storm. it 's expected to make landfall sunday on the southeastern coast of [UNK] province .
Summary by model [2]	tropical storm maysak approached the asian island nation saturday . it's now classified as a tropical storm , according to the philippine national weather service . it's now classified as a tropical storm , according to the philippine weather service .
Summary by model [3]	just a few days ago , maysak gained super typhoon status thanks to its sustained 150 mph winds . it 's now classified as a tropical storm , according to the philippine national weather service .
Summary by model [4]	super typhoon could weaken . new jersey , but it will . philippine ocean strength . at least 132 people are injured , including 18 .

Table 2: Comparison of the generated summary using the summarization models v.s. the ground truth

[2] has repeated a sentence twice. The summary generated by the pointer-generator together with the coverage mechanism not only could have overcome the $< unk >$ problem but also does not have repetition in the generated summary and gives a nice summary pretty close to the reference summary. The summary generated by the transformer model can only capture some keywords but does not convey the grasp of summary very well.

Fake News Detection Subsequent to Summarization

In this part, we use the best summarization model that we have trained on the summarization dataset in order to create summaries of a fake news detection dataset. We will build a fake news detection model and we investigate its performance when the input is the original news text, the news headline, and the summarized news text generated by our summarization model. Basically, We use our text summarizing model as a feature generator for a fake news classification model. In fake news classification the article content contains much more information than the article headline and due to this a fake news classifier performs better on article contents than on article headlines.
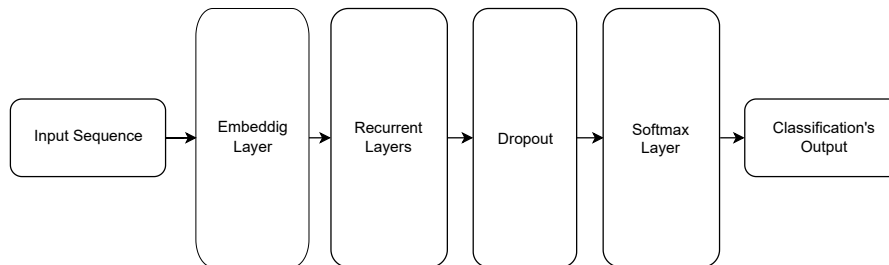

Figure 5: Fake news classification architecture

For fake news classification, we use a fake news dataset with headlines and article content provided by George McIntire [1]. The dataset contains 3164 fake news articles and 3171 real articles (i.e. a balanced dataset) on politics from a wide range of news sources. We shuffle the data and use 80% of it for training, 10% of it for validation, and 10% for testing, and also do 5-fold cross validation.
We build a Long-short-term-memory (LSTM) network together with an Embedding Layer as shown in Figure 5. Table 3 shows our hyperparameter studies for fake news classification and Table 4 shows the final test accuracies, using the three input features of full body text, headline text, and generated summary texts by our summarization models. As it can be seen in this table the best model using the body text as input features perform better than headline text as input. Furthermore, it's worth noting that the summary text as input feature leads to an even higher accuracy compared to the full body text as input feature. This finding shows that summarization model serves as a feature generator for fake

---

[1]https://www.datasciencecentral.com/profiles/blogs/on-building-a-fake-news-classification-model
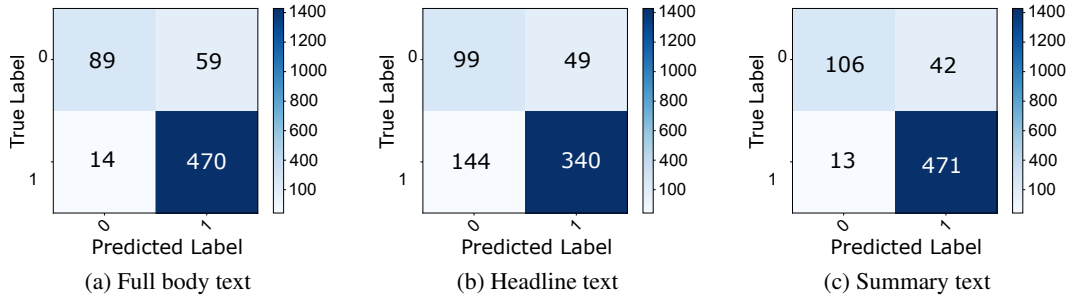
| (a) Full body text | (b) Headline text | (c) Summary text |

Figure 6: Confusion matrix for test set of fake news detection task using three different input features

Input features	Exp. #	Cells	Size	Drop-out	train loss	train acc. %	valid loss	valid acc. %
Full body text	1	**LSTM**	**64**	**0.2**	0.081	97.4	0.12	**92.1**
	2	LSTM	64	0.5	0.143	93.7	0.167	91.5
	3	LSTM	128	0.2	0.076	97.2	0.178	91.4
	4	LSTM	128	0.5	0.141	94.5	0.224	90.2
	5	Bi-LSTM	64	0.2	0.025	99.1	0.129	90.9
	6	Bi-LSTM	64	0.5	0.08	97.1	0.128	91.2
	7	Bi-LSTM	128	0.2	0.026	99.2	0.111	89.7
	8	Bi-LSTM	128	0.5	0.0741	97.3	0.113	88.6
Headline text	1	LSTM	64	0.2	0.121	95.3	0.241	91.6
	2	LSTM	64	0.5	0.157	93.2	0.215	91.3
	3	LSTM	128	0.2	0.099	95.9	0.227	91.7
	4	LSTM	128	0.5	0.156	93.6	0.221	91.0
	5	**Bi-LSTM**	**64**	**0.2**	0.103	95.6	0.229	**91.8**
	6	Bi-LSTM	64	0.5	0.154	93.5	0.219	91.0
	7	Bi-LSTM	128	0.2	0.106	95.7	0.239	91.5
	8	Bi-LSTM	128	0.5	0.158	93.5	0.217	91.1
Summary text	1	LSTM	64	0.2	0.074	97.3	0.291	92.1
	2	LSTM	64	0.5	0.146	94.5	0.231	92.2
	3	LSTM	128	0.2	0.083	97	0.247	92.3
	4	LSTM	128	0.5	0.139	94.6	0.201	91.3
	5	Bi-LSTM	64	0.2	0.078	97.1	0.291	91.9
	6	Bi-LSTM	64	0.5	0.152	94.1	0.246	91.6
	7	**Bi-LSTM**	**128**	**0.2**	0.079	97.1	0.221	**93.1**
	8	Bi-LSTM	128	0.5	0.146	94.5	0.242	91.8

Table 3: Experiments on the fake news detection

Input Features	Accuracy %	Average Length (in words)
Full body text	92	10.51
Headline text	91	387.89
Summary text	93	20.41

Table 4: Fake news classifier results

news detection task which actually increases its accuracy. Also, this summarization model can also serve as a headline generator for the news articles as an automatic approach.

4 Conclusion

As we showed in section 3.3 the pointer-generator architecture with attention and coverage mechanisms led to the highest accuracies and could overcome the problems common in abstractive text summarization such as out-of-vocabulary words and repetition. Furthermore, as shown in section 3.3 a text summarizing model can successfully be applied as a feature generator prior to classification tasks such as fake news classification and increase the accuracy of those tasks.

Code References

Summarization

https://github.com/abisee

https://github.com/ymfa

https://github.com/hashbangCoder

https://github.com/atulkum

Transformer

https://github.com/tensorflow/models/tree/master/official/transformer

https://github.com/JRC1995/Transformers

https://github.com/gooppe/

https://github.com/lipiji

Literature References

Mehdi Allahyari, Elizabeth D Trippe, and Juan B Gutierrez. Text Summarization Techniques : A Brief Survey.

Bonnie Dorr, David Zajic, and Richard Schwartz. Hedge Trimmer: A Parse-and-Trim Approach to Headline Generation. (March 2004):1–8, 2007. doi: 10.3115/1119467.1119468.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. SummaRuNNer: A Recurrent Neural Network based Sequence Model for Extractive Summarization of Documents. (c):3075–3081, 2016a. ISSN 02561115. doi: 10.1007/s11814-017-0328-2. URL http://arxiv.org/abs/1611.04230.

Chandra Khatri, Gyanit Singh, and Nish Parikh. Abstractive and Extractive Text Summarization using Document Context Vector and Recurrent Neural Networks. 2018. URL http://arxiv.org/abs/1807.08000.

Shen Gao, Xiuying Chen, Piji Li, Zhaochun Ren, Lidong Bing, Dongyan Zhao, and Rui Yan. Abstractive Text Summarization by Incorporating Reader Comments. 2018. URL http://arxiv.org/abs/1812.05407.

Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. ISSN 12864579.

Ramesh Nallapati, Bing Xiang, and Bowen Zhou. SEQUENCE-TO-SEQUENCE RNNS FOR TEXT SUMMA-RIZATION. pages 4–7, 2016b.

Karl Moritz, Hermann Tom, and Will Kay. Teaching Machines to Read and Comprehend NIPS 2015. pages 1–9.

Abigail See, Peter J Liu, and Christopher D Manning. Get To The Point: Summarization with Pointer-Generator Networks. 2017. ISSN 15420752. doi: 10.18653/v1/P17-1099. URL http://arxiv.org/abs/1704.04368.

Kiyoharu J. Miyagishima, Ulrike Grünert, and Wei Li. Processing of S-cone signals in the inner plexiform layer of the mammalian retina, 2014. ISSN 14698714. URL http://arxiv.org/abs/1706.03762.

Zhaopeng Tu and Zhengdong Lu. Modeling Coverage for Neural Machine Translation. pages 76–85, 2016.

Danqi Chen, Jason Bolton, and Christopher D Manning. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task.

William Yang Wang. WikiHow: A Large Scale Text Summarization Dataset.

Chin-yew Lin and Marina Rey. ROUGE : A Package for Automatic Evaluation of Summaries. 2001.